AVR112: TWI Bootloader for devices without boot section

8-bit Atmel Microcontrollers

Application Note

Features

- All 8-bit Atmel AVR microcontrollers with
  - 2K bytes Flash
  - Without boot section
  - Having two wire TWI /USI Interface
- AVROSP compatible
- Example application code for all supported target devices
- Using IAR workbench for debugging and programming
- C-Code Driver for TWI Master
- C-Code Driver for TWI / USI Slave
- Application Section Programming
  - Without any programming tool
  - Application section complete erase
  - Application section programming
  - Application software version readability
  - Bootloader software version readability

1 Introduction

This application note describes the implementation of Bootloader using TWI or USI hardware as TWI communication channel for 8-bit Atmel® AVR® microcontrollers, without on-chip boot section and with at least 2Kbytes of flash memory. The host device is based on ATmega2560 communicating via the UART with a PC running the AVR Open Source Programmer (AVROSP). . Host device will act as TWI Master while the Target will act as TWI Slave. The host device then communicates with target device via TWI.

This enables the user to download Application code into the target mcu, to read boot loader version and application version details without having an external programmer.

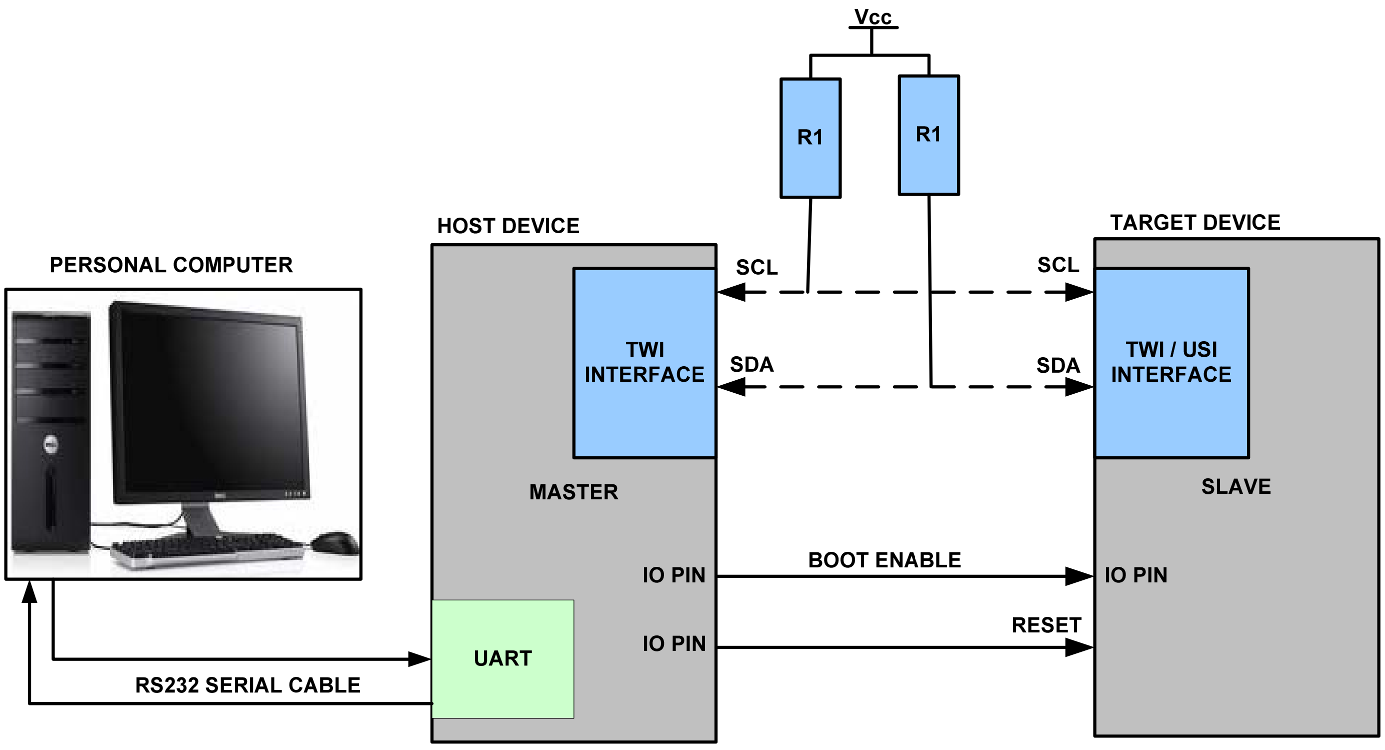
The downloaded file AVR112.zip contains all utilities and examples necessary to build a test application, program it into Slave device and verifying the test application.

For general information about self programming please refer to application note AVR109: Self programming.

The hardware interfacing diagram is illustrated in Figure 1-1.

Rev. 8079A-AVR-08/11

Figure 1-1. PC, Host and Target device interface diagram



2 Theory

This section gives a detailed description of the microcontroller bootloader. For more detailed information on TWI / USI module, refer the datasheets.

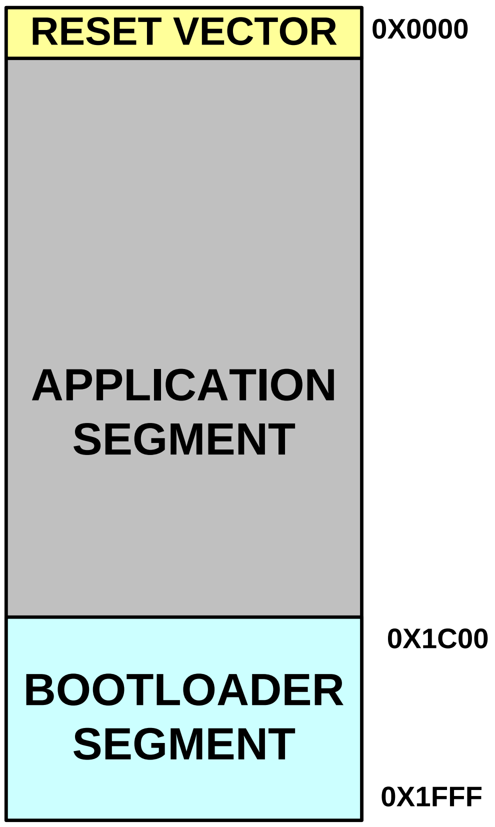
2.1 What is Microcontroller Boot loader and how it works?

In general, to program a microcontroller, one needs a programmer that supports the target device. Apart from being expensive, it might have other disadvantages, like long programming times, too many interconnections between the microcontroller and the programmer itself. Upgrading the firmware for a shipped product becomes tedious because of non availability of programming port. Bootloader comes handy, when there are standard communications interfaces being available on a connector. Even in the laboratories, it might not be practical to get a programmer for each working bench, and the best solution is to provide microcontroller samples which have already been programmed once with the Bootloader in the program memory.

Once bootloader is programmed, the respective interface (UART or TWI) can be used to reprogram the remaining flash section without using a conventional programmer. In this application note, the bootloader uses the TWI interface and other pins to detect the upgrade process. First the Bootloader firmware checks for the BOOT PIN status, and if found low, then the program in the configured boot section starts executing, else the program in the configured Application section starts executing.

The Figure 2-1 gives a brief idea on the flash section which is divided between Application and Boot section.

Figure 2-1. Flash Memory Map



Please note the addresses specified in the figure: 2-1, varies with different target devices depending on their flash.

Summarizing, the steps involved a Bootloader are:

Step 1 – After reset, the Bootloader checks for the Boot Pin Status, and depending on the condition code flow works in Bootloader or jumps to Application. When in boot section, the Bootloader code starts to listen the TWI interface commands for incoming data.

Step 2 – If the command received is for updating the flash, the received data is written in the program memory (application section) as it would be written by a programmer.

Step 3 – Once the entire incoming data has been written and again if the command received is to execute the application code, the Bootloader executes a jump at the first instructions of the regular program.

2.2 Bootloader Requirements

The TWI Bootloader makes it unnecessary for any physical intervention with a hardware programmer. The target device must have some form of data connection with the host. In this application note, IIC compatible TWI communication is used. If the device uses the TWI port to communicate with other devices on the bus, then no additional connection is needed. The Bootloader will require the higher side of the program memory reserved for the Bootloader firmware (see Figure 2-1). The rest of the program memory can be used by application. No RAM memory needs to be reserved, since it is used in different contexts and not at the same time.

2.3 BASIC OPERATION

The basic functionality of a Bootloader is to receive, interpret and execute a set of commands known both by the host software and the target firmware. The command set can be more or less complex than shown in the application note, but it must generally support commands to read, erase and write the Flash memory. Additionally, it supports a command to jump from the Bootloader code to the application code. This command is typically issued at the end of a program cycle.

3 Hardware Setup

This section describes the role of a TWI Master and TWI Slave, their configuration and its physical connectivity.

3.1 TWI Master – Description

TWI master accepts the commands from the AVROSP PC software through UART. The received command will be sent across to the TWI Slave via TWI communication channel. In this application note, ATmega2560 device on STK600 development board is used as TWI master and the connection details are mentioned in the table 3-1.

3.2 TWI Slave – Description

TWI slave is target device where the application section to be programmed via bootloader. The TWI slave accepts the commands via TWI communication channel and performs the predefined activities on the target device. In this application note, ATtiny88 device is used as target device on STK600 development board.

3.3 TWI Master and TWI Slave configuration

This section describes how the TWI master and TWI slaves are configured with target voltage clock frequency, fuse settings etc.

3.3.1 TWI Master Configuration

Device: ATmega2560

Target Voltage: 5V

Target Frequency: External Crystal 3.69MHz

Fuse Settings: 0XFF (Extended) 0X19 (High) 0XFD (Low)

3.3.2 TWI Slave Configuration

The slave could be any device from the below list. The slave devices are categorized into two lists based on the availability of the peripheral module. Some have built in TWI and others have USI hardware which can be used as TWI interface. Section "**Devices with TWI interface**" and "**Devices with USI interface**" briefs on the devices which fall in the respective ones. The USI hardware is configured as TWI slave, for more details please refer the application note: "AVR312: Using the USI module as an I2C slave". As an example, ATtiny88 configuration is described in "ATtiny88 as TWI Slave". For all other devices in the below list, please refer "Hardware Setup" page and particular device column of Configuration Details.xls excel sheet document in the downloaded zip folder.

3.3.2.1 Devices with TWI interface:

ATtiny48, ATtiny88, ATmega48A, ATmega48PA.

3.3.2.2 Devices with USI interface:

ATtiny45-45V, ATtiny85-85V, ATtiny24A, ATtiny44A, ATtiny84-84V, ATtiny2313A, ATtiny4313, ATtiny261A, ATtiny461A, ATtiny861A, ATtiny43U, ATtiny87, ATtiny167.

3.3.2.3 ATtiny88 as TWI Slave:

Device: ATtiny88

Target Voltage: 5V

Target Frequency: 8MHz Internal RC Oscillator

Fuse Settings: 0XFE (Extended) 0XDD (High) 0XEE (Low)

3.4 Hardware connection between TWI Master and TWI Slave

The test setup includes two STK600, one with TWI master and the second with TWI Slave. For all the supported target devices, the connection details between Master and Slave is described in the "Hardware Setup" page of Configuration Details.xls excel sheet document in the downloaded zip folder. As an example, the connection details for TWI master (ATmega2560) and TWI Slave (ATtiny88) is mentioned below table 3-1.

Table 3-1. Hardware connection between TWI master and TWI Slave device

ATMEGA2560 as TWI MASTER	ATTINY88 as TWI SLAVE	Description
M_PD1_SDA	S_PC4_SDA	Master SDA line, M_PD1_SDA is connected to the Slave SDA line, S_PC4_SDA (TWI SDA line must be pulled-up externally)
M_PD0_SCL	S_PC5_SCL	[TB9L - Table Body 9pt_Left]
M_PC0_Enable	S_PB0	This pin helps to Enable Bootloader, active low, polled by Bootloader when it starts. Master M_PC0_Enable pin is connected to the Slave S_PB0 pin
M_PC1_RESET	S_PC6/RESET	Master resets the slave using this line, note: disable debugWire on slave. Master M_PC1_RESET pin is connected to the S_PC6/RESET pin
VCC	VCC	Both master and Slave has to connected with Vcc
GND	GND	Common Ground for both Mater and Slave
	LED0_PB1	LED0 is Connected on Slave PB1 pin
	LED1_PB3	LED1 is Connected on Slave PB3 pin
RS232 SPARE to PC Serial (don't use USB to Serial Converter)		Connect Master device serial port to PC, to accept the AVROSP commands from PC

4 Software Setup

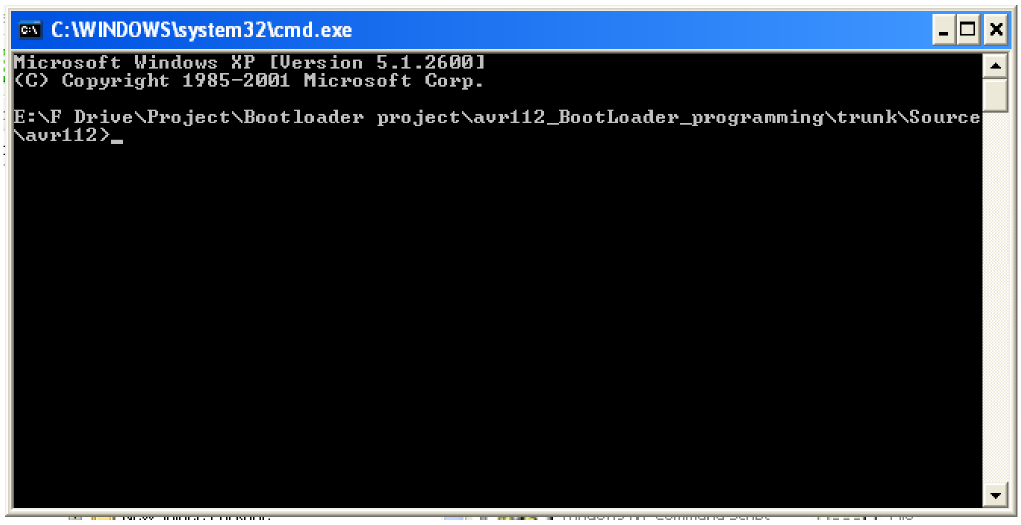
This section gives a detailed description of how to use the PC Software, TWI_Master firmware, TWI Slave firmware and Demo Application Software. It is must to have the master programmed with “” and slave with "TWI_Slave.a90" files using external programmer in order to perform the up gradation of slave application section.

The "TWI_Slave.a90" program responds to the commands from “TWI_Master”, and the “TWI_Master” will be getting commands from PC via AVROSP Windows® software through UART and allows the chip to reprogram.

4.1 PC Side Software

Here, we use AVROSP as PC software and ATmega2560 is configured as TWI master. The PC software, AVROSP has the primary function of importing the hex file (which is to be programmed in Slave device), splitting it into smaller packets and sending the data to the master device via RS232 interface.

Figure 4-1. AVROSP software application window



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

E:\F Drive\Project\Bootloader project\avr112_BootLoader_programming\trunk\Source
\avr112>_
```

For more details on AVROSP, please refer Application note “AVR911: AVR Open Source Programmer”

4.1.1 BOOTLOADER COMMANDS

The Bootloader firmware used in this design supports a set of commands used to read, erase and write to the Flash memory of the target device. The commands are shown in Table 4-1.

Table 4-1. Bootloader Commands

Command Name	Description
TWIBL COMSETUP	Configures COM PORT for AVROSP communication The command prompt should display the following: Enter the COM_PORT_NUMBER 1 Status for device COM1: ----------------------- Baud: 115200 Parity: None Data Bits: 8 Stop Bits: 1 Timeout: ON XON/XOFF: OFF CTS handshaking: OFF DSR handshaking: OFF DSR sensitivity: OFF DTR circuit: ON RTS circuit: ON
TWIBL HOSTSIG	Reads the Signature of TWI_Master, to ensure communication between TWIBL_PROG (Software) and TWI_MASTER
TWIBL UPDATE <filename.hex>	Loads Application (<filename.hex>) to TWI Slave via Boot loader.
TWIBL DIAG	Displays diagnostic messages for last command except for TWI
TWIBL EXEC	Jumps to Application section from Bootloader and run Application
TWIBL ERASE	Erase entire application section except RESET Vector
TWIBL BVERSION	Read Bootloader Version
TWIBL AVERSION	Read Application Version

4.2 TWI Master Firmware

The Master firmware is written to accept the commands and slave's application hex file from the PC side AVROSP software. After receiving the desired packets, it is sent to the slave device via TWI and helps to perform the flash upgrade on the TWI Slave device.

This firmware is responsible to communicate to both PC and Target device using UART and TWI respectively.

4.3 TWI Slave Firmware

The TWI Slave firmware is developed for all the AVR microcontrollers with at least 2Kbyte of flash memory, without boot section and communication channel, is either TWI or USI hardware as TWI. This bootloader is responsible to receive the flash content from TWI master (ATmega2560) and perform the necessary actions depending on the commands received.

5 Getting Started

1. Get the hardware connection ready mentioned in the “**MASTER PROGRAMMER**” column of Hardware Setup of **Configuration Details.xls** excel sheet document in the downloaded zip folder.

5.1 Programming the TWI Master Device

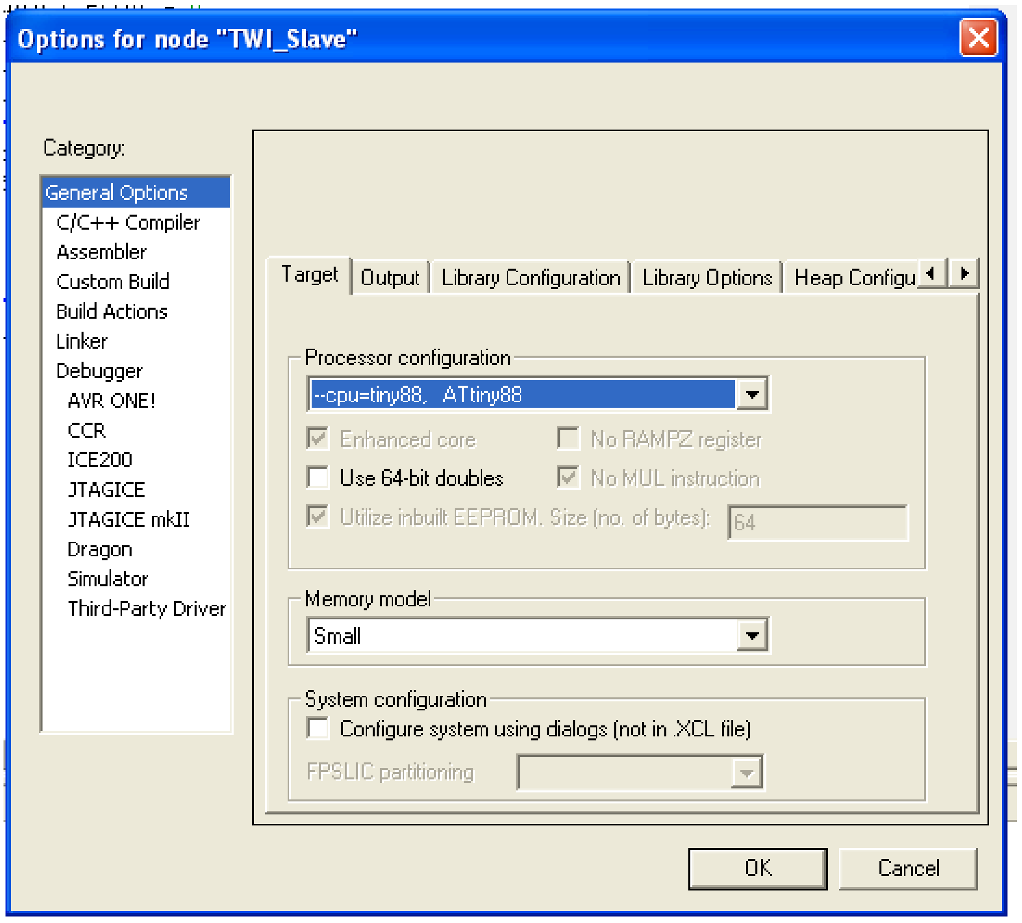
1. Open \avr112_BootLoader_programming\trunk\Source\avr112\TWI_Master\TWI_Master.eww
2. Save All
3. Clean
4. Rebuild all
5. Program ATmega2560 with …TWI_Master\Release\Exe\TWI_Master.a90

5.2 Programming the TWI Slave Device

1. Open \avr112_BootLoader_programming\trunk\Source\avr112\TWI_Master\TWI_Slave.eww
2. Select the TWI Slave device in IAR Embedded Workbench® IDE as mentioned in the below two figures. For instance here ATtiny88 device is selected as TWI slave.

Go to Project → Options→ Category→General Options→ Target tab

Figure 5-1. IAR Embedded Workbench IDE Setup1



Processor configuration: --cpu=tiny88, ATtiny88

- [✓] Enhanced core
- [ ] No RAMPZ register
- [ ] Use 64-bit doubles
- [✓] No MUL instruction
- [✓] Utilize inbuilt EEPROM. Size (no. of bytes): 64

Memory model: Small

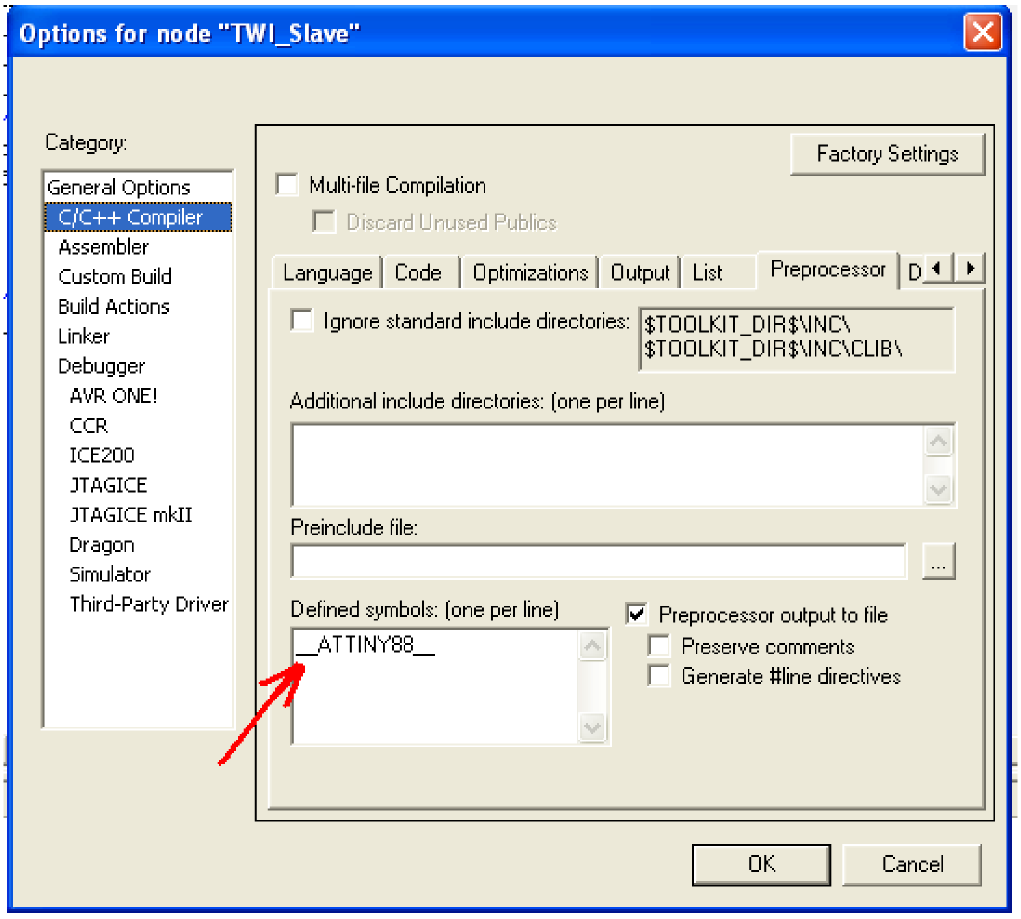
System configuration

- [ ] Configure system using dialogs (not in .XCL file)

FPSLIC partitioning

Project → Options→ Category→ C/C++ Compiler→ Preprocessor tab

Figure 5-2. IAR Embedded Workbench IDE Setup2


Options for node "TWI_Slave"

Category:

General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
AVR ONE!
CCR
ICE200
JTAGICE
JTAGICE mkII
Dragon
Simulator
Third-Party Driver

Factory Settings

☐ Multi-file Compilation
☐ Discard Unused Publics

Language | Code | Optimizations | Output | List | Preprocessor | D

☐ Ignore standard include directories: $TOOLKIT_DIR$\INC\ $TOOLKIT_DIR$\INC\CLIB\

Additional include directories: (one per line)

Preinclude file:

Defined symbols: (one per line)
__ATTINY88__

☑ Preprocessor output to file
☐ Preserve comments
☐ Generate #line directives

OK | Cancel

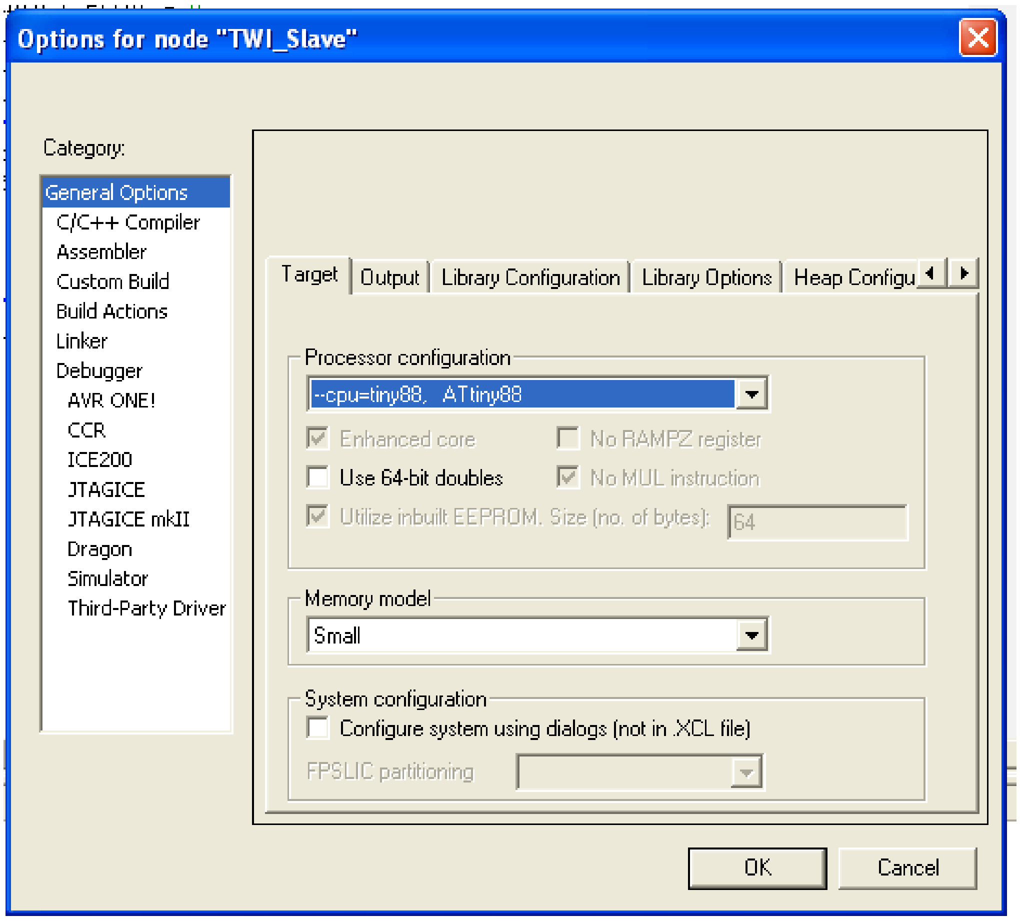
3. Save All
4. Clean
5. Rebuild all
6. Program Selected Slave device with …TWI_Slave\Release\Exe\TWI_Slave.a90

5.3 Generate hex file for LED blink Application to be programmed via Bootloader

1. Open \avr112_BootLoader_programming\trunk\Source\avr112\Demo_App1\Demo_App1.eww
2. Select the TWI Slave device in IAR Embedded Workbench IDE as mentioned in the below two figures. For instance here ATtiny88 device selected as TWI slave.

Go to Project → Options→ Category→General Options→ Target tab

Figure 5-3. IAR Embedded Workbench IDE Setup1


Options for node "TWI_Slave"

Category:

General Options
C/C++ Compiler
Assembler
Custom Build
Build Actions
Linker
Debugger
AVR ONE!
CCR
ICE200
JTAGICE
JTAGICE mkII
Dragon
Simulator
Third-Party Driver

Target | Output | Library Configuration | Library Options | Heap Configu

Processor configuration
--cpu=tiny88, ATtiny88
[✓] Enhanced core
[ ] No RAMPZ register
[ ] Use 64-bit doubles
[✓] No MUL instruction
[✓] Utilize inbuilt EEPROM. Size (no. of bytes): 64

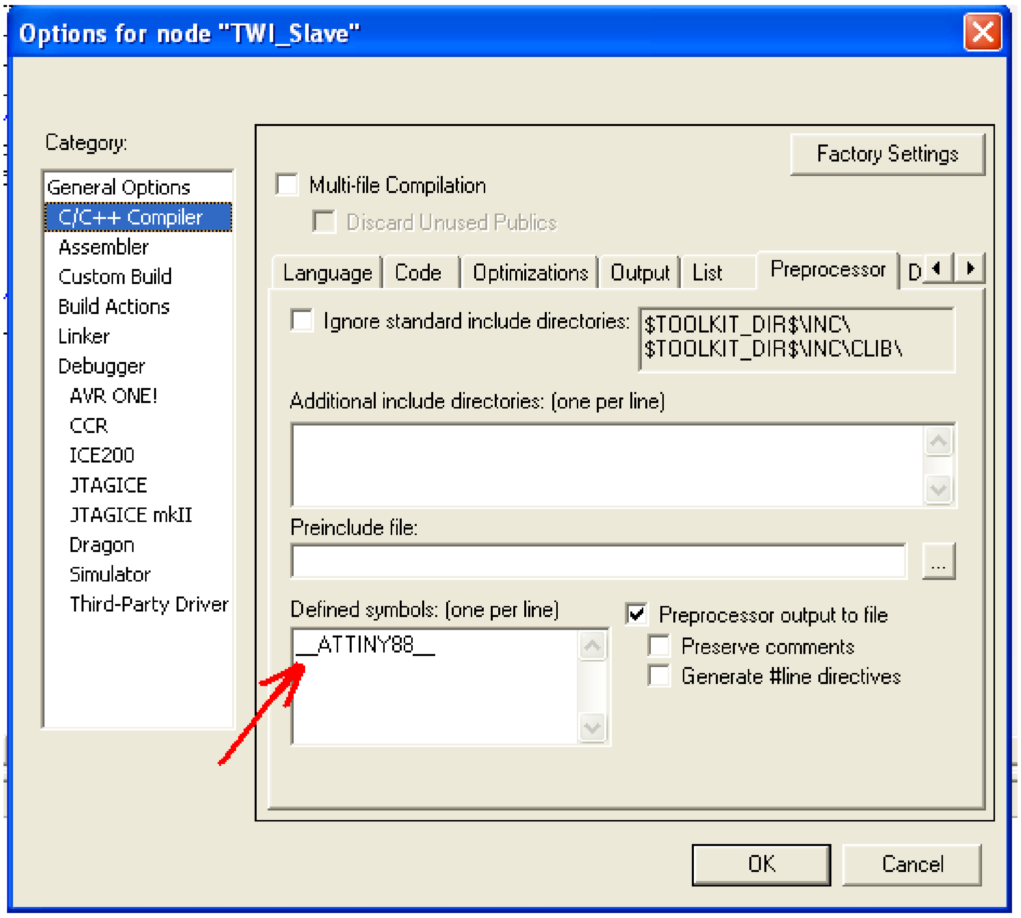
Memory model
Small

System configuration
[ ] Configure system using dialogs (not in .XCL file)
FPSLIC partitioning

OK | Cancel

Project → Options→ Category→ C/C++ Compiler→ Preprocessor tab

Figure 5-4. IAR Embedded Workbench IDE Setup2


Options for node "TWI_Slave"

Category: C/C++ Compiler

- [ ] Multi-file Compilation
- [ ] Ignore standard include directories: $TOOLKIT_DIR$\INC\ $TOOLKIT_DIR$\INC\CLIB\
- Additional include directories: (one per line)
- Preinclude file:
- Defined symbols: (one per line) __ATTINY88__
- [✓] Preprocessor output to file
- [ ] Preserve comments
- [ ] Generate #line directives

3. Save All
4. Clean
5. Rebuild all
6. DemoApp1.hex is now ready

5.4 Functionality Test

1. Power both TWI Master and TWI Slave (Both LED0 and LED1 will be LIT to indicate Host boot delay).
2. Double-Click file “ Start.Cmd “
3. Type TWIBL , It will ask for COM PORT NUMBER for the first time, enter the COM PORT NUMBER that is connected to the Master. This will configure the COMPORT settings. This will also list the TWIBL commands.
4. Type TWIBL UPDATE DemoApp1.hex to program the application section with Demo_App1.hex.
5. Type TWIBL EXEC to run the programmed code. If target Slave is programmed with DemoApp1.hex successfully, this will blink LED0.
6. Type TWIBL ERASE to erase the programmed application.

7. Type TWIBL AVERSION to get Application Version from EEPROM address 0X05. What happens if application sections are not programmed?
8. Type TWIBL BVERSION to get Bootloader Version.

Similarly, generate another application hex file to blink LED1 and reprogram the TWI Slave device using the above mentioned procedure.

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: (+1)(408) 441-0311
Fax: (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Milennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
Tel: (+852) 2245-6100
Fax: (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parkring 4
D-85748 Garching b. Munich
GERMANY
Tel: (+49) 89-31970-0
Fax: (+49) 89-3194621

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chou-ku, Tokyo 104-0033
JAPAN
Tel: (+81) 3523-3551
Fax: (+81) 3523-7581